LR parser

هو احد المعربات الاكثر شيوعا من معربات Bottom-Up حيث ان :

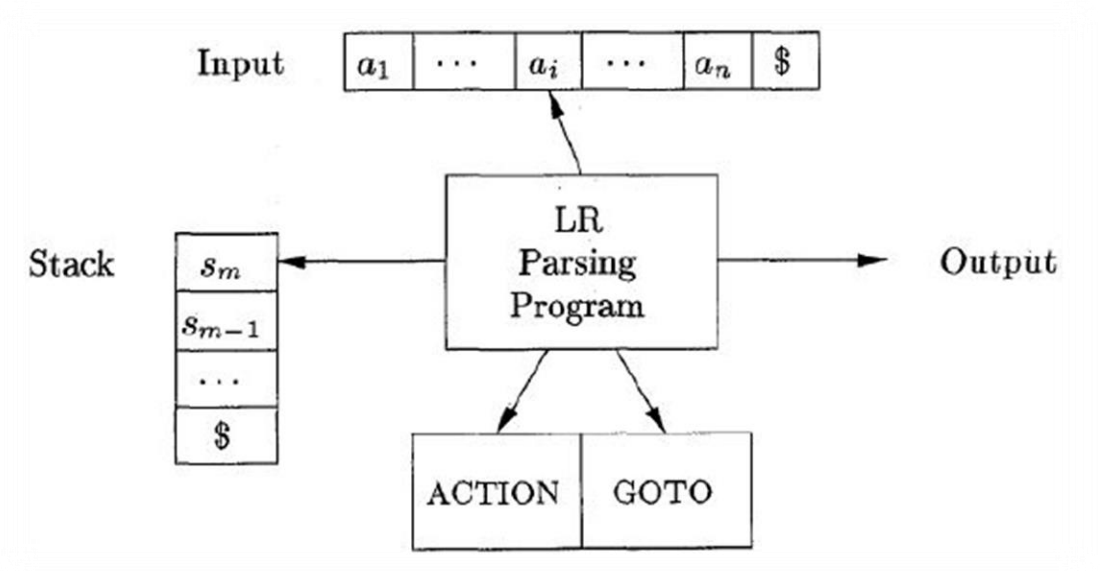
L: stands for Left to Right Scan

R: stands for Right Most Derivation in Reverse

K: number of symbols for look ahead (k<=1)

يتألف هذا المعرب من ربع اجزاء :

1- **Input**
2- **Output**
3- **Stack**
4- **Driver program**
5- **Parsing table that has two parts : ACTION and GOTO**



LR algorithm

Set cursor to the right most symbol of w$

Push initial state s_0 on top of stack

Repeat

Let s be the state on top of stack

Let a be the current pointed symbol in w$

If action[s,a]=shift s' then

Push a on top of stack

Push s' on top of stack

Advance cursor to the next symbol on the right in w$

Else if action[s , a]=reduce $A \rightarrow \beta$ then

Pop $2|\beta|$ symbols of the stack

Let s' be the state on top of stack

Push A on top of stack

Push goto[s', A] on top of stack

Output $A \rightarrow \beta$

Else if action[s , a]=accept then

Return

Else error

The algorithm in otherform

```
token = next_token()
repeat forever
s = top of stack
if action[s, token] = "shift si" then
    PUSH token
    PUSH si
    token = next_token()
else if action[s, token] = "reduce A::= β" then
    POP 2 * |β| symbols
    s = top of stack
    PUSH A
    PUSH goto[s,A]
else if action[s, token] = "accept" then
    return
else
    error()
```

Example : suppose the following CFG and the table

1) $E \rightarrow E + T$
2) $E \rightarrow T$
3) $T \rightarrow T * F$
4) $T \rightarrow F$
5) $F \rightarrow (E)$
6) $F \rightarrow id$

state	Action						Goto		
	id	+	*	(	)	$	E	T	F
0	S5			S4			1	2	3
1		S6				Acc			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

Let us check the string w = id * id + id

Stack	*Input*	*Action*
0	*id * id + id $*	*shift*
0 id 5	** id + id $*	*Reduce F---> id*
0 F 3	** id + id $*	*Reduce T---> F*
0 T 2	** id + id $*	*shift*
*0 T 2 * 7*	*id + id $*	*shift*
*0 T 2 * 7 id 5*	*+ id $*	*Reduce F---> id*
*0 T 2 * 7 F 10*	*+ id $*	*Reduce T---> T*F*
0 T 2	*+ id $*	*Reduce E--->T*
0 E 1	*+ id $*	*shift*
0 E 1 + 6	*id $*	*shift*
0 E 1 + 6 id 5	*$*	*Reduce F---> id*
0 E 1 + 6 F 3	*$*	*Reduce T---> F*
0 E 1 + 6 T 9	*$*	*Reduce E--->E+T*
0 E 1	*$*	*Acc*